UNIX essentials (hands-on)

- the directory tree

- running programs

- the shell (using the T-shell)

  → command line processing

  → special characters

  → command types

  → shell variables

  → environment variables

  → wildcards

  → shell scripts

  → shell commands

  → pipes and redirection

- OS commands

- special files

- **The Directory Tree**

    → directories contain files and/or directories

    → **/** : means either the root directory, or a directory separator

    - consider **/home/afniuser/AFNI_data3   afniuser/suma_demo**

        ➢ (perhaps **/Users/macuser_42/AFNI_data3** on a mac)

    → an "absolute" pathname begins with '**/**', a "relative" pathname does not

    - a relative pathname depends on where you start from

    - in the directories above, note which is a relative pathname

    → every directory has a parent directory

    - the relative pathname for the parent directory is '..'

    - the relative pathname for the current directory is '.'

    - consider '**./run_this_script**' and '**/bin/ls ../../suma_demo**'

    → many commands can be used to return to the home directory (of "**afniuser**")

    - **cd, cd ~, cd ~afniuser, cd $HOME, cd /home/afniuser**

        ➢ on a mac **$HOME** might be **/Users/macuser_42**

        ➢ note the 2 special characters, '~' and '**$**'

    → while you work, keep your location within the directory tree in mind

→ class work:

- open a terminal window
- commands: **cd, pwd, ls, ls -al**
- use the "**cd**" command to go to the given directories
    - ➢ e.g. for directory **/usr/bin**, use the command: **cd /usr/bin**
    - ➢ once there, use the commands "**pwd**", "**ls**", and "**ls -al**"
    - ➢ note that you can always return to the home directory via: **cd**

AFNI_data3   ..   AFNI_data3/afni   ..   ~/abin

- first example (starting with the '**AFNI_data3** 'directory), use the commands:

  ```
  cd AFNI_data3
  pwd
  ls
  ls -al
  ```

- **Running Programs**
  - → a program is something that gets "executed", or "run"
  - → the first element of a command line is generally a program (followed by a space)
  - → most shells are case sensitive when processing a command
  - → command examples:
    - ➢ `/bin/ls $HOME ~/AFNI_data3`
    - ➢ `count -digits 2 1 10`
  - → script: an interpreted program (interpreted by another program)
    - e.g. shell script, javascript, perl script, afni startup script
    - view the regression script: `cat ~/AFNI_data3/afni/epi_r1_regress`
  - → some commands: `cd, pwd, echo, ls, wc, cat, less, nedit, man`
    - ➢ `cd      ~/AFNI_data4`   – change directories
    - ➢ `wc      s2.anova`   – word count
    - ➢ `cat     s2.anova`   – concatenate (to terminal)
    - ➢ `less    s2.anova`   – a text file perusal program
    - ➢ `gedit   s2.anova`   – a GNU text editor
    - ➢ `man     wc`   – an online manual (runs in `less` mode)

    - basic keystrokes for `less` (and `man`): *`Enter, Space,`* `b, g, G, h, q`

- **The Shell**
  - → command interpreter (case and syntax sensitive)
  - → examples: **tcsh, csh, sh, bash, ksh, zsh, wish, tclsh, rsh, ssh**
  - → command: **echo $SHELL**
  - → the T-shell**: /bin/tcsh**
    - • an enhanced C-shell (**csh**), which has C programming style syntax
- **Command Line Processing** (simplified outline):
  - 1) evaluate special characters, such as: ~ **$ & * ? \ ' " ` |**
  - 2) decide which program to execute (more on this later)
    - • pathname, alias, shell command, search the **$PATH**
  - 3) execute appropriate program, passing to it the parameter list
  - 4) save the execution status in the **$status** variable (0 is considered success)
  - → command: **echo hello $HOME '$pickle'**
  - → tcsh has automatic filename completion using the Tab key
    - • type "**ls suma**" and hit the *Tab* key, watch what happens, and hit *Enter*
    - • type "**ls AF**" and hit the *Tab* key, note what happens

- **Special Characters**

  ~ : the current user's home directory (e.g. /home/user), same as $HOME

  $: used to access a variable (e.g. $home)

  & : used to put a command in the background (e.g. afni &)

  * : wildcard, matching zero or more characters (e.g. ls AFNI_d*)

  ? : wildcard, matching exactly one character (e.g. ls AFNI_data?)

  **** : command line continuation (must be the last character on the line)

  ' : the shell will not evaluate special characters contained within these quotes

     (e.g. echo '**$HOME**' → will output **$HOME**, not **/home/afniuser**)

     (e.g. **3dbucket -prefix small_func 'func_slim+orig[0,2..4]'**)

  " : the shell will evaluate **$variables** and **`commands`** contained within these

     (e.g. **echo "[*] my home dir is $HOME"**)

     (e.g. **echo "the numbers are 'count -digits 2 7 12'"**)

  ` : execute the command contained within these quotes, and replace the quoted part with the output of the contained command

     (e.g. **echo "the numbers are `count -digits 2 7 12`"**)

- **Command Types**
  - → the shell must decide what type of command it has:
    - pathname for a program: execute that program
    - alias: apply any alias(es) then start over (decide on which program to run)
    - shell command: part of the `/bin/tcsh` program
    - check the `$PATH` directories for the program

  - → consider the commands:

    ```
    /bin/ls AFNI_data3/afni
    ls AFNI_data3/afni
    cd AFNI_data3/afni
    wc ~/AFNI_data3/afni/epi_r1_ideal.1D
    ```
  - → the "which" command shows where the shell gets a command from:

    ```
    which ls            which cd            which wc
    ```
- **The PATH Variable**
  - → a list of directories to be searched for a given program to be run from
  - → the `$path` and `$PATH` variables are identical, but are represented differently
  - → commands: `echo $PATH`

    ```
              echo $path
              cat ~/.cshrc
    ```

- **Shell Variables**
    - → shell variables are variables that are stored in, and affect the shell
    - → all variables are stored as strings (or as arrays of strings)
    - → a variable is accessed via the '**$**' character
    - → the '**echo**' command: echo the line after processing any special characters
        - command: **echo my home dir, $HOME, holds ~/***
    - → the '**set**' command: set or assign values to one or more variables
        - without arguments: '**set**' displays all variables, along with any values
        - '**set**' takes a list of variables to set, possibly with values
        - consider the commands:

```
set food
echo $food
set food = pickle
echo $food
set food eat = chocolate donut      (emphasis: food  eat = chocolate  donut)
set
set food = eat chocolate donut
set food = "eat chocolate donut"
echo $food
```

→ variables can be assigned the result of a numerical computation using the '@' command, however only integer arithmetic is allowed

- commands: `set value1 = 17`

   `@ value2 = $value1 * 2 + 6`

   `echo value2 = $value2`

- **Array Variables**
  - → array variables are set using ()
  - → consider the commands:

    ```
    set stuff = ( 11 12 13 seven 15 )
    echo $stuff
    echo $stuff[1]
    echo $stuff[2-4]
    echo $stuff[8]
    set stuff = ( hi $stuff $food )
    echo $stuff
    echo $path
    cat ~/.cshrc
    ```

- **Environment Variables**

  → similar to shell variables, but their values will propagate to children shells

  → by convention, these variables are all upper-case (though it is not required)

  → similarly, shell variables are generally all lower-case

  → set environment variables using "`setenv`" (as opposed to the "`set`" command)

  → without any parameters, the "`setenv`" command will display all variables

  → the "`setenv`" command will only set or assign one variable at a time

  → the format for the command to set a value is (without any '=' sign):

  ```
  setenv VARIABLE value
  ```

  - commands:
    ```
    setenv MY_NAME Elvis
    echo $MY_NAME
    echo $path
    echo $PATH
    echo $HOME
    setenv
    ```

- **Wildcards**
    - → used for shell-attempted filename matching
    - → special characters for wildcards:

        *, ?, [,], ^

        * : matches any string of zero or more characters

            (special case: a lone * will not match files starting with '.')

        ? : matches exactly one character

        [ ] : matches any single character within the square brackets

        [^] : matches any single character EXCEPT for those within the brackets

    - → commands (run from the **~/AFNI_data3/MPRAGE_anat** directory):

        ```
        ls
        ls *
        ls -a
        ls M*3.dcm
        ls M*0*3.dcm
        ls M*0?3.dcm
        ls M*[23]*.dcm
        ls M*[^23]*.dcm
        echo M*[^23]*.dcm
        ```

- **Shell Scripts**
  - → a text file, a sequence of shell commands
  - → the '\' character can be used for line continuation (for readability)
    - • for that purpose, it <u>must</u> be the last character on the line (including spaces)
  - → executing shell scripts, 3 methods:
    - **1) ./filename**   : (safest) execute according to the top "**#!program**"
      - ➤ if no such line, usually executed via **bash** (a potential programming error)
      - ➤ the file must have execute permissions (see '**ls -l**')
    - **2) tcsh filename**   : execute as t-shell commands
    - **3) source filename** : execute using current shell
      - ➤ affects current environment
      - ➤ this method should be used only when that is the intention (e.g. **.cshrc**)
  - → consider **~/AFNI_data4/s1.afni_proc.block**
  - → consider **~/AFNI_data3/afni/rall_regress**
  - → use the command "**gedit my.script**" to create a script with a few commands
    - **echo hi, I am in directory $cwd**
    - **ls -a**
    - **cd $HOME/AFNI_data3**
    - **ls -al**
  - → run the script using the command: **tcsh my.script**

- **Some Shell Commands** (handled by the shell)

`cd`	: change working directory
`echo`	: echo command line to the terminal window
`pwd`	: display the present working directory
`set`	: set variables or assign string values to variables
`@`	: set a variable to the results of an integral computation
`alias`	: display or create an alias
	(e.g. `alias hi 'echo hello there'`)
`bg`	: put a process in the background (usually after ctrl-z)
`fg`	: put a process in the foreground
`exit`	: terminate the shell
`setenv`	: set environment variables
`source`	: execute a script within the current shell environment

- special keystrokes (to use while a process is running)

`ctrl-c`	: send an interrupt signal to the current process
`ctrl-z`	: send a suspend signal to the current process

- **More Shell Commands: basic flow control**
  - → commands: `if, else, endif, while, end, foreach`

```
if ( $user == "elvis" ) then
     echo 'the king lives'
endif
```

```
set value = 5
set fact = 1
while ( $value > 0 )
   @ fact = $fact * $value
   @ value -= 1
end
echo 5 factorial = $fact
```

```
foreach value ( 1 2 3 four eight 11 )
   echo the current value is $value
end
```

```
foreach file ( I.*3 )
   ls -l $file
end
```

- **Pipes and Redirection**
    - **>** : redirect program output (stdout) to a file

        e.g. **`3dmerge -help > 3dmerge.help`**

        `3dmerge -pickle > 3dmerge.help`
    - **>&** : redirect all output (both **`stdout`** and **`stderr`**) to a file

        e.g. **`3dmerge -pickle >& 3dmerge.pickle`**

        e.g. **`tcsh my.script >& script.output`**

    - **>>** : append program output to a file

        e.g. **`echo "more info: value = $val" >> script.output`**

    - **|** : pipe standard output to the input of another program

        e.g. **`3dDeconvolve -help | less`**

    - **|&** : include **`stderr`** in the pipe

        e.g. **`tcsh my.big.script  |&  tee script.output`**
        - · run the script
        - · send all output to the tee program
        - · the tee program duplicates the input, sending the output to both the terminal and the given file (**`script.output`**)
        - · you can see the output, but it is also stored for future analysis

- **Some OS Commands**

  `ls`   : list the contents of a directory

  * `cat`   : concatenate files to the terminal (print them to the screen)

  * `more`   : a file perusal program - view files one page at a time

  * `less`   : a better file perusal program (type `less`, get `more`)

  `man`   : on-line manuals for many OS commands (and library functions)

        - this uses a "`less`" interface to display the information

        - e.g. consider `man` on : `ls, less, man, tcsh, afni`

  * `head`   : display the top lines of a file (default = 10)

        - e.g. `3dDeconvolve -help | head -25`

  * `tail`   : display the bottom lines of a file (default = 10)

        - e.g. `tail ideal_r1.1D`

  * `wc`   : word count - count characters, words and lines (of a file)

  `cp`   : copy files and directories to a new location

  `mv`   : rename a file, or move files and direcotories

  `rm`   : BE CAREFUL - remove files and/or directories (no recovery)

        - e.g. `rm junk.file`

        - e.g. `rm -r bad.directory`

*   denotes a 'filter' program, which can take input from a file or from `stdin`

* **grep** : print lines from a file that match the given pattern

> e.g. **grep path ~/.cshrc**

> e.g. **ls ~/abin | grep -i vol**

> e.g. from the output of "**3dVol2Surf -help**" show lines which
> contain 'surf', but not 'surface', then remove duplicates:

3dVol2Surf -help | grep surf | grep -v surface | sort | uniq

• Some Special Files (in the home directory)

.cshrc : c-shell startup file ("**csh** run commands")

> ➢ set aliases

> ➢ adjust the path

> ➢ set shell and environment variables

.afnirc : **AFNI** startup file

.sumarc : **suma** startup file

.login : commands run at the start of a login shell (e.g. a terminal window)

.logout : commands run before exiting a login shell

.tcshrc : t-shell startup file (if it does not exist, the .cshrc file will be used)